SPI Slave with Single Port RAM

Project Description can be found in this link.

- Create Tcl file to create the vivado project and run the full design flow on Vivado using the Tcl file
- Create an XDC file where the rst_n , SS_n & MOSI are connected to 3 switches, and the MISO to a led.
- The SPI slave implementation is done using FSM, we shall try out three different encoding (gray, one_hot or seq) using the following vivado attribute in your Verilog code

  (* fsm_encoding = "gray" *)

- Snippet:

```
(* fsm_encoding = "gray" *)
reg [1:0] cs, ns;

always @(posedge clk or posedge rst) begin
if (rst)
    cs <= S0;
else
    cs <= ns;
end
```

- This attribute **must** be inserted just above your declared current and next states (cs, ns)
- We wish to operate at the highest frequency possible and so you shall choose the encoding based on the best timing report that gives the high setup slack after implementation.
- After choosing the best encoding, add a debug core such that all internals (MISO, MOSI, SS_n, rst_n & clk) can be analyzed and then generate a bitstream file

Deliverables:

>> I will use the do file and Tcl file written from your side to run the simulation on QuestaSim and to run the design flow on Vivado so make sure that they are correct and functionable. <<

1) The assignment should be submitted as a PDF file with this format <your_name>_Project2 for example Kareem_Waseem_Project2
2) Write in the PDF file the group member names. **Please use the same certificate names here**.

I am expecting a **rar** file having the design files, testbench file, do file, XDC file, Tcl file, netlist generated by the Tcl file, bitstream file and a PDF file.

The PDF file will have snippets of the following:

1) Snippets from the waveforms captured from QuestaSim for the design with inputs assigned values and output values visible.
2) Synthesis snippets for each encoding
   - Schematic after the elaboration & synthesis
   - Synthesis report showing the encoding used
   - Timing report snippet
   - Snippet of the critical path highlighted in the schematic
3) Implementation snippets for each encoding
   - Utilization report
   - Timing report snippet
   - FPGA device snippet
4) Snippet of the "Messages" tab showing no critical warnings or errors after running elaboration, synthesis, implementation and a successful bitstream generation.

Project

- Project team
  - — Teams of three
- Final Project Report
  - — Deadline: TBD in the lecture
  - — Submit the following
    - – PDF file with snippets
    - – Design files
    - – One Testbench file
    - – Do file to run the testbench
  - — Allow 2 days for unforeseen issues

Project Suggestions

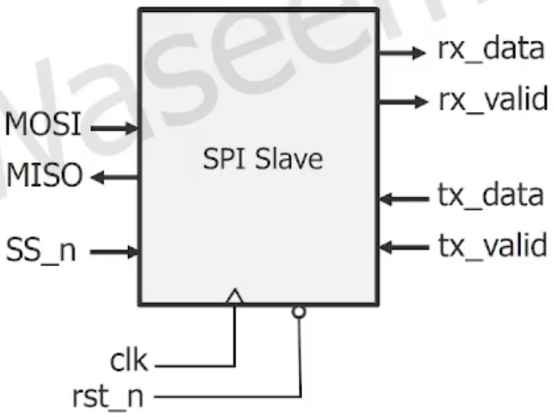
- Split the work in parallel between design and verification
- Tasks will take longer than what you think
- Have a 2-3-day task list

SPI Interface

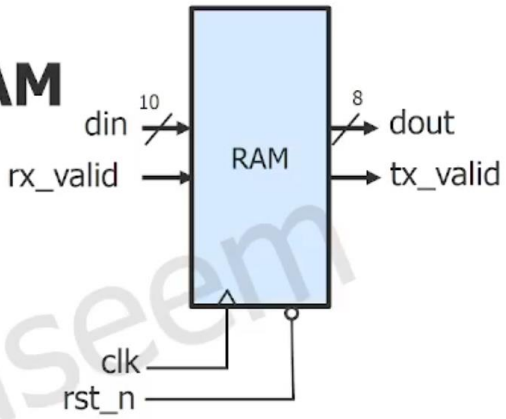
- One of the most popular Interfaces nowadays
- Stands for Serial-Peripheral Interface
- Four Wires
  - MOSI: Master-Out-Slave-In
  - MISO: Master-In-Slave-Out
  - SCK: Clock
  - SS_n: Slave Select
- High Data Rates

Project: 1- SPI Slave Interface

- One of the most popular Interfaces nowadays
- Stands for Serial-Peripheral Interface
- Four Wires
  - MOSI: Master-Out-Slave-In
  - MISO: Master-In-Slave-Out
  - SCK: Clock
  - SS_n: Slave Select
- High Data Rates



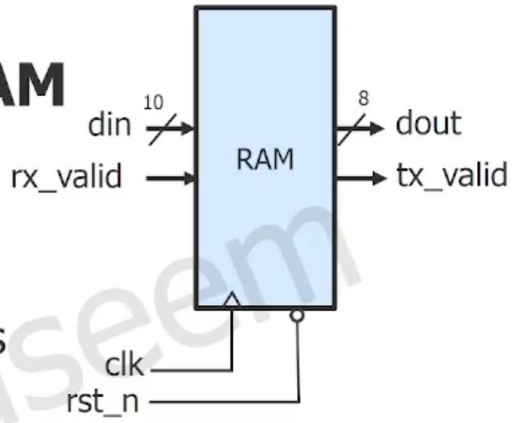
Project: 2- Single-port Sync RAM



- **Parameters**
  - MEM_DEPTH, Default: 256
  - ADDR_SIZE, Default: 8
- **Ports**

Name	Type	Size	Description
din	Input	10 bits	Data Input
clk		1 bit	Clock
rst_n		1 bit	Active low asynchronous reset
rx_valid		1 bit	If HIGH: accept din[7:0] to save the write/read address internally or write a memory word depending on the most significant 2 bits din[9:8]
dout	Output	8 bits	Data Output
tx_valid		1 bit	Whenever the command is memory read the tx_valid should be HIGH

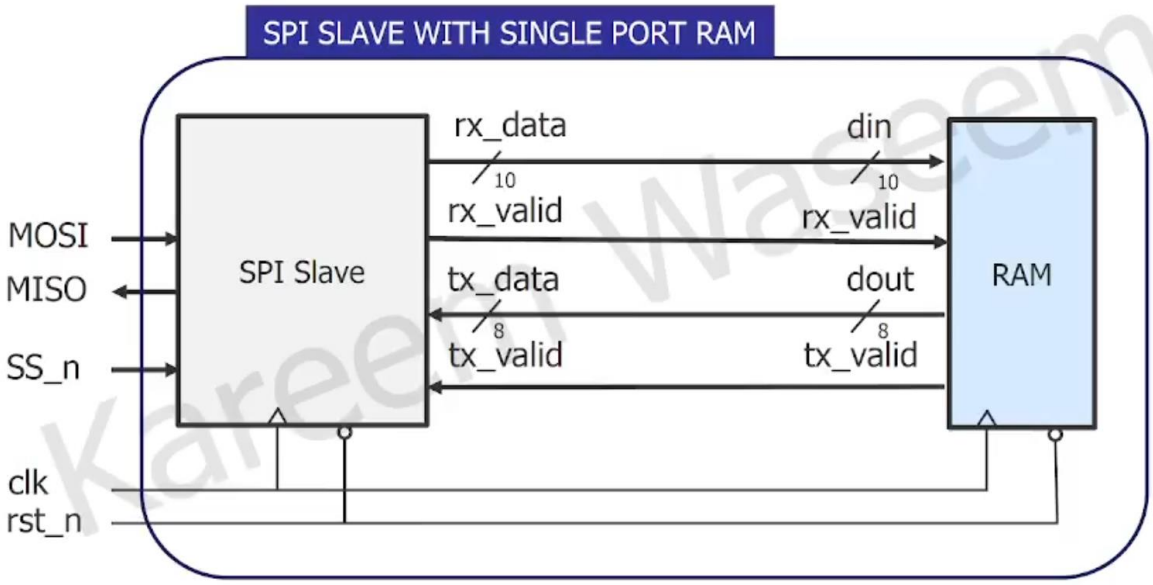
Project: 2- Single-port Sync RAM

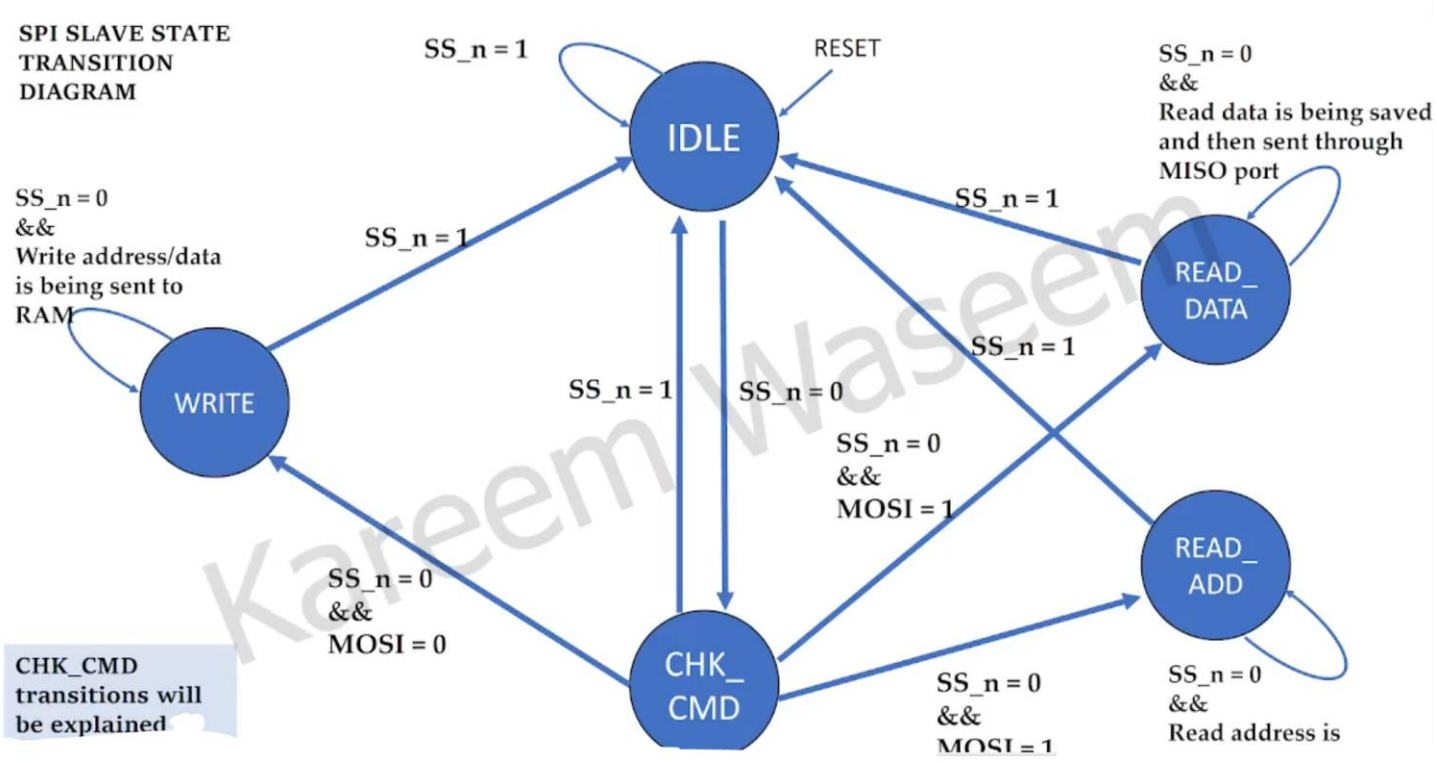


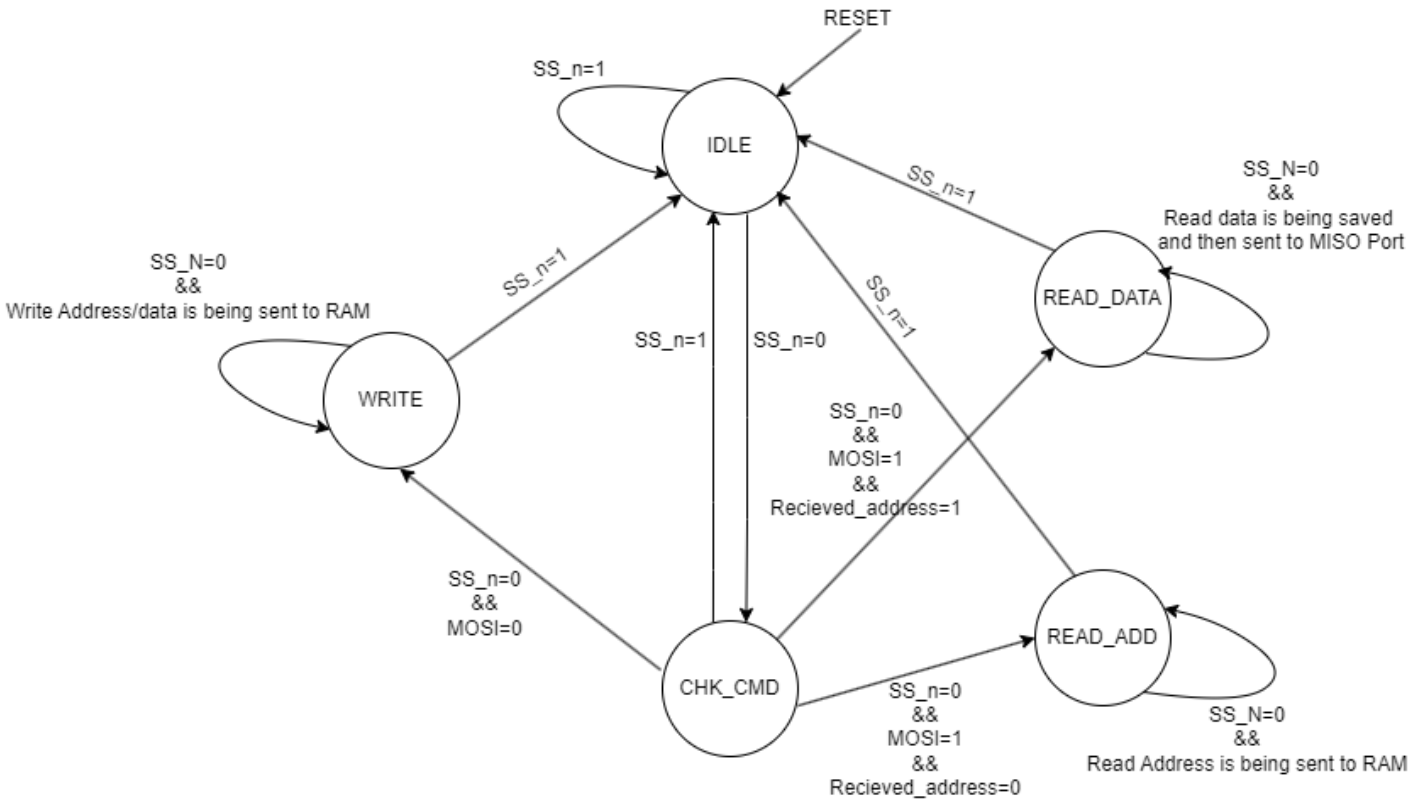
- **Parameters**
  - MEM_DEPTH, Default: 256
  - ADDR_SIZE, Default: 8
- Most significant din bit "din[9]" determines if it is a write or read command

Port	Din[9:8]	Command	Description
din	00	Write	Hold din[7:0] internally as write address
	01		Write din[7:0] in the memory with write address held previously
	10	Read	Hold din[7:0] internally as read address
	11		Read the memory with read address held previously, tx_valid should be HIGH, dout holds the word read from the memory, ignore din[7:0]

Project: 3- SPI Wrapper

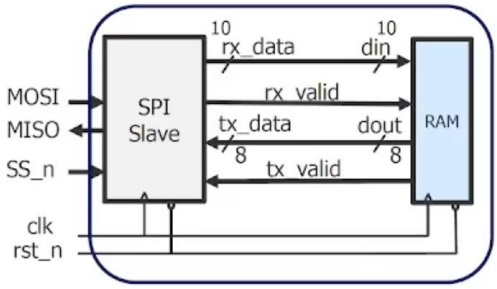




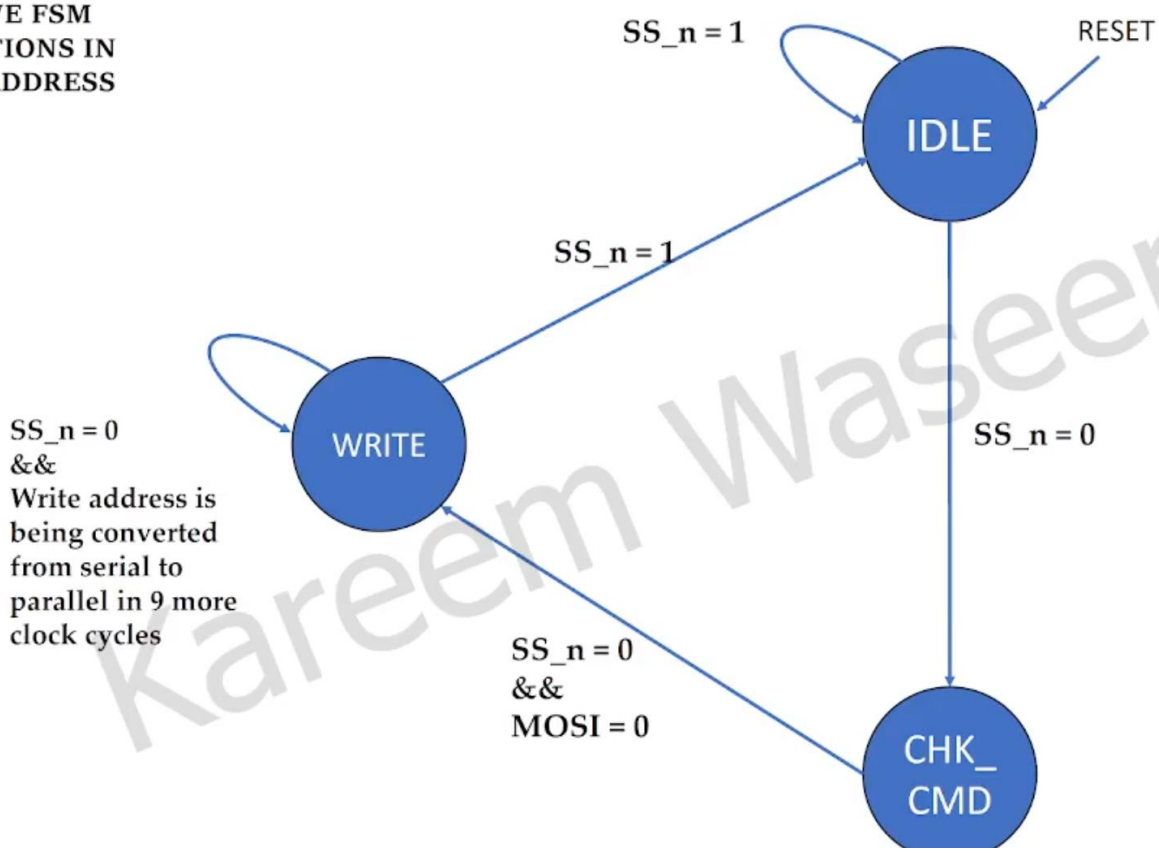


RAM Write Command – Write Address

1. Master will start the write command by sending the write address value, rx_data[9:8] = din[9:8] = 2'b00
2. SS_n = 0 to tell the SPI Slave that the master will begin communication
3. SPI Slave check the first received bit on MOSI port '0' which is a control bit to let the slave determine which operation will take place "write in this case". SPI Slave then expects to receive 10 more bits, the first 2 bits are "00" on two clock cycles and then the wr_address will be sent on 8 more clock cycles
4. Now the data is converted from serial "MOSI" to parallel after writing the rx_data[9:0] bus
5. rx_valid will be HIGH to inform the RAM that it should expect data on din bus
6. din takes the value of rx_data
7. RAM checks on din[9:8] and find that they hold "00"
8. RAM stores din[7:0] in the internal write address bus
9. SS_n = 1 to end communication from Master side

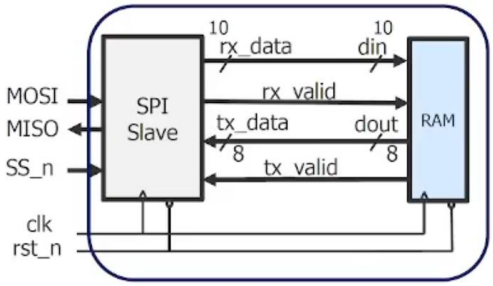


SPI SLAVE FSM TRANSITIONS IN WRITE ADDRESS

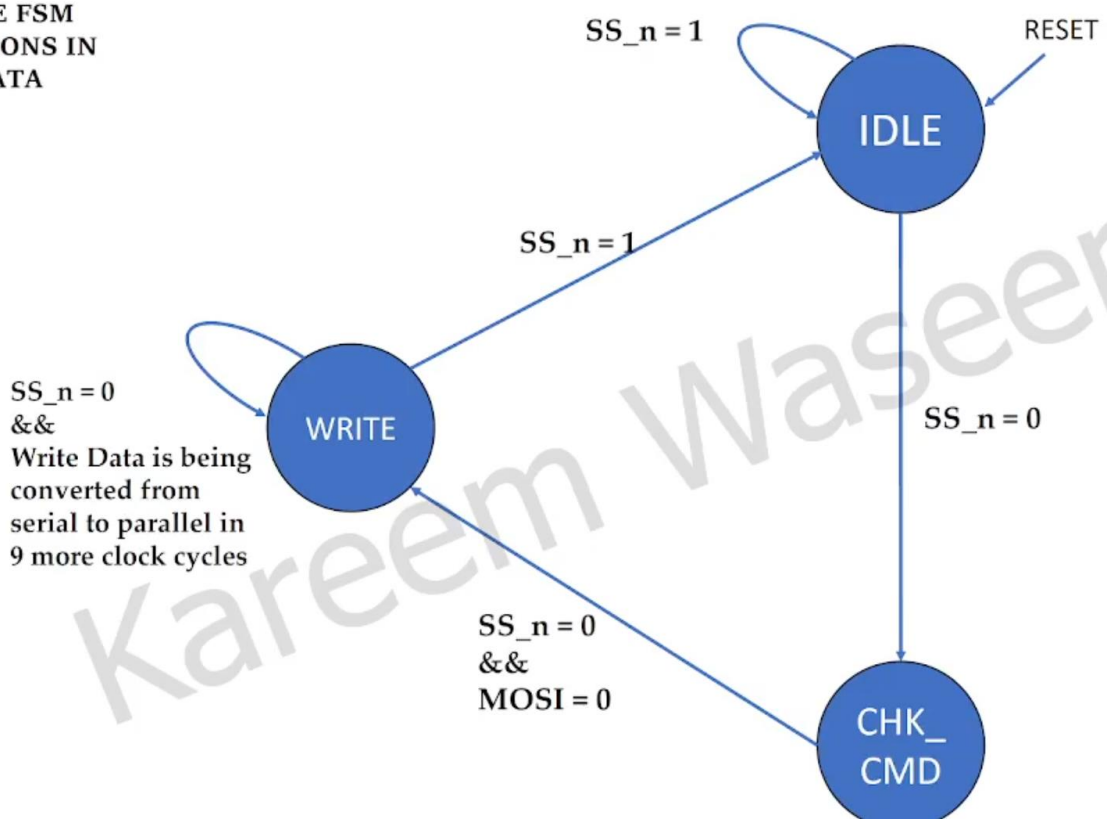


RAM Write Command – Write Data

1. Master will continue the write command by sending the write data value, rx_data[9:8] = din[9:8] = **2'b01**
2. SS_n = 0 to tell the SPI Slave that the master will begin communication
3. SPI Slave check the first received bit on MOSI port '0' which is a control bit to let the slave determine which operation will take place "write in this case". SPI Slave then expects to receive 10 more bits, the first 2 bits are "01" on two clock cycles and then the wr_**data** will be sent on 8 more clock cycles
4. Now the data is converted from serial "MOSI" to parallel after writing the rx_data[9:0] bus
5. rx_valid will be HIGH to inform the RAM that it should expect data on din bus
6. din takes the value of rx_data
7. RAM checks on din[9:8] and find that they hold **"01"**
8. RAM stores din[7:0] in the RAM with wr_address previously held
9. SS_n = 1 to end communication from Master side

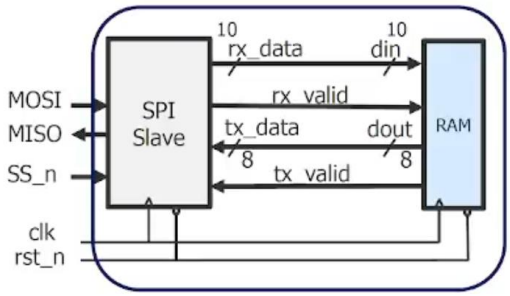


SPI SLAVE FSM TRANSITIONS IN WRITE DATA

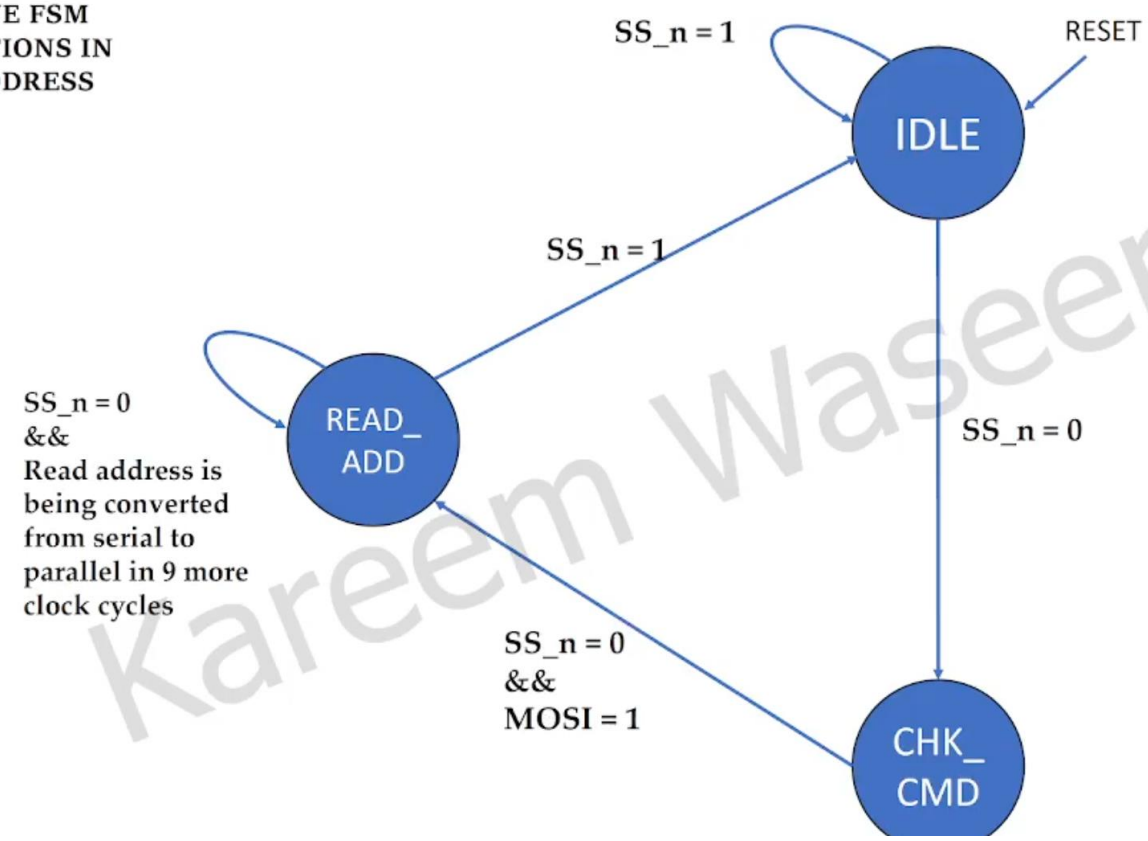


RAM Read Command – Read Address

1. Master will start the write command by sending the read address value, rx_data[9:8] = din[9:8] = 2'b10
2. SS_n = 0 to tell the SPI Slave that the master will begin communication
3. SPI Slave check the first received bit on MOSI port '1' which is a control bit to let the slave determine which operation will take place "read in this case". SPI Slave then expects to receive 10 more bits, the first 2 bits are "10" on two clock cycles and then the rd_address will be sent on 8 more clock cycles
4. Now the data is converted from serial "MOSI" to parallel after writing the rx_data[9:0] bus
5. rx_valid will be HIGH to inform the RAM that it should expect data on din bus
6. din takes the value of rx_data
7. RAM checks on din[9:8] and find that they hold "10"
8. RAM stores din[7:0] in the internal read address bus
9. SS_n = 1 to end communication from Master side

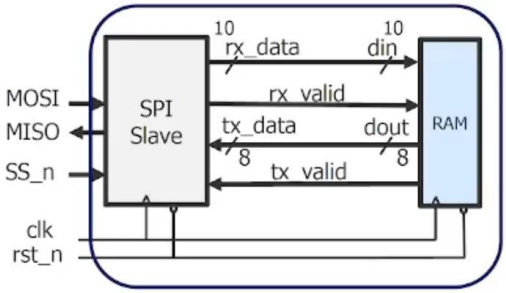


SPI SLAVE FSM TRANSITIONS IN READ ADDRESS

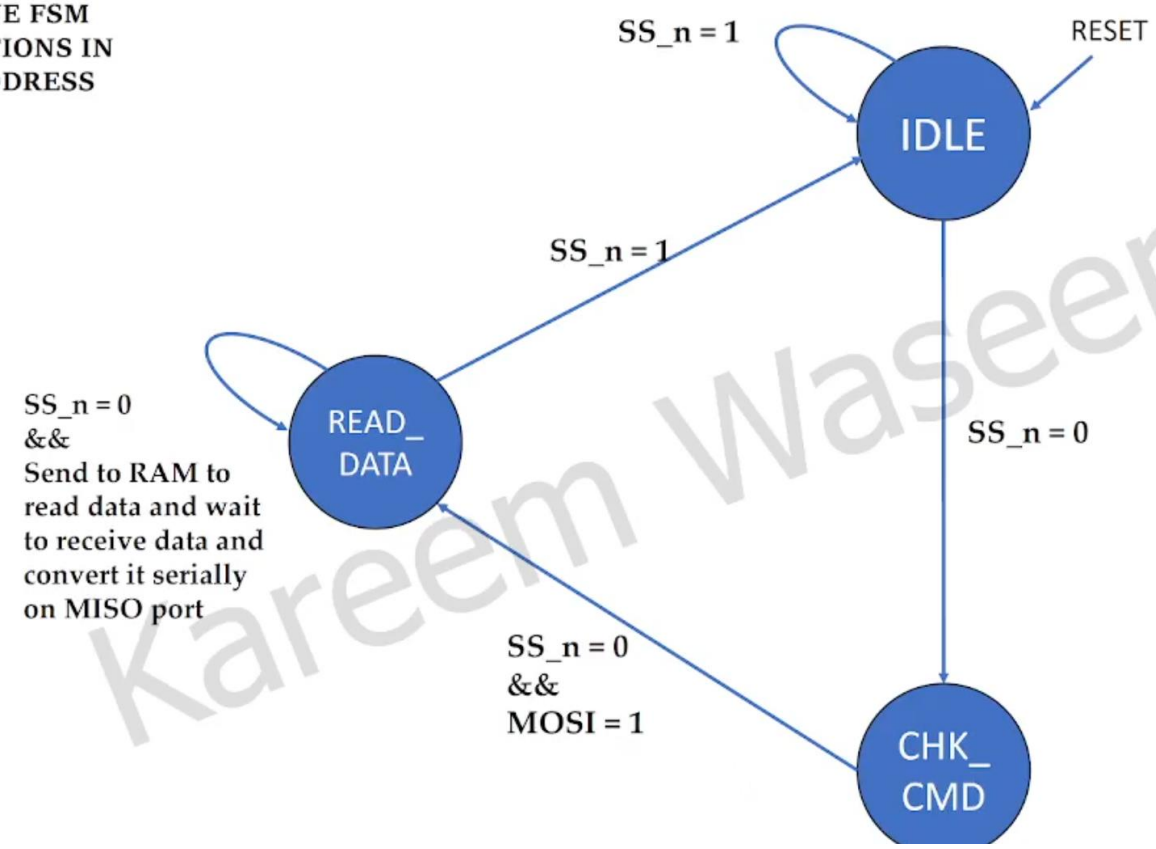


RAM Read Command – Read Data

1. Master will start the write command by sending the read address value, rx_data[9:8] = din[9:8] = 2'b1**1**
2. SS_n = 0 to tell the SPI Slave that the master will begin communication
3. SPI Slave check the first received bit on MOSI port '1' which is a control bit to let the slave determine which operation will take place "read in this case". SPI Slave then expects to receive 10 more bits, the first 2 bits are "11" on two clock cycles and then dummy data will be sent and ignored since the master is waiting for the data to be sent from slave side
4. Now the data is converted from serial "MOSI" to parallel after writing the rx_data[9:0] bus
5. din takes the value of rx_data
6. RAM reads din[9:8] and find that they hold "11"
7. RAM will read from the memory with rd_address previously held
8. RAM will assert tx_valid to inform slave that data out is ready
9. Slave reads tx_data and convert it into serial out data on MISO port
10. SS_n = 1, Master ends communication after receiving data "8 clock cycles"



SPI SLAVE FSM TRANSITIONS IN READ ADDRESS



SPI Slave implementation Suggestions

- Split the SPI Slave design into three **always** blocks
  - State Memory always block
  - Next state Logic always block
  - Output Logic always block with posedge clk as the sensitivity
- Extra signals that you will need
  1. Counter for serial to parallel and vice verse Conversion
  2. Internal Signal to allow SPI slave to memorize if the read address is received or not to decide for READ_ADD or READ_DATA transition